Supporting Adaptive Services in a Heterogeneous Mobile Environment

Nigel Davies, Gordon S. Blair, Keith Cheverst and Adrian Friday

Distributed Multimedia Research Group,
Department of Computing,
Lancaster University,
Bailrigg,
Lancaster,
LA1 4YR,
U.K.

telephone: +44 (0)524 65201
e-mail: nigel, gordon, kc, adrian@comp.lancs.ac.uk

Abstract

Future computer environments will include mobile computers which will either be disconnected, weakly inter-connected by low speed wireless networks such as GSM, or fully inter-connected by high speed networks ranging from Ethernet to ATM. While the transition between networks is currently a heavyweight operation we believe that developments in network interface technology will soon enable mobile computers to dynamically select their network service based on cost and performance requirements. Such flexibility, coupled with the inherent unreliability of mobile communications, means that system services and applications will be subject to rapid and massive fluctuations in the quality-of-service provided by their underlying communications infrastructure. In this paper we discuss the design of a distributed systems platform to support the development of services which are able to tolerate this environment by dynamically adapting to changes in the available communications quality-of-service.

1: Introduction

Future computer environments will include mobile computers which will either be disconnected, weakly inter-connected by low speed wireless networks such as GSM, or fully inter-connected by high speed networks ranging from Ethernet to ATM ([1, 2]). Two key characteristics of such environments are:-

(i) a heterogeneous processing environment (including relatively low-power mobile hosts) and,

(ii) rapid and massive fluctuations in the *quality of service* (QoS) provided by the underlying communications infrastructure.

The first of these issues, i.e. heterogeneity, can be addressed by exploiting emerging distributed systems standards such as the International Standards Organisation Reference Model for Open Distributed Processing (RM-ODP) [3], the Open Software Foundation's Distributed Computing Environment (OSF-DCE) [4] or the Object Management Group's Common Object Request Broker Architecture (OMG-CORBA) [5]. Such standards provide applications with uniform computational models for accessing services and enables them to operate over a variety of processor/operating system configurations. The second of these issues, QoS fluctuations, is, we believe, one of the most fundamental problems in the field of mobile computing. In this position paper we address this issue and propose extensions to emerging distributed systems standards to support mobile computing. These extensions are designed to allow the development of *adaptive or reactive applications* [2, 6] which are able to tailor their behaviour based on changes in their communications infrastructure.

Section 2 of this paper presents a rational for developing adaptive services and highlights the role of QoS management in supporting such services. Section 3 then describes a distributed systems platform we have developed based on the ODP-like ANSAware system [7]. This platform provides QoS management facilities to enable the development of adaptive services and applications in a heterogeneous environment. The platform also supports the transmission of continuous media types (e.g. voice). Section 4 describes a prototype adaptive application we are developing at Lancaster based on the platform described in section 3. Finally, section 5 contains our concluding remarks.

2: Adaptive Services

2.1: Analysis of the Problem

As mentioned above, we believe that the key characteristic of mobile computing environments is that end systems experience differing degrees of connectivity during typical operational cycles. In particular, systems are expected to function when connected to a range of networks including high-speed networks (fully connected operation) and low-speed wireless networks (weakly connected

0-8186-6345-6/95 $04.00

operation), and when totally disconnected (disconnected operation). To date, the transition between modes of operation has been a heavy-weight process often involving both hardware and software re-configuration. However, the advent of technologies such as the MINT mobile internet router[1] promise to significantly reduce the overheads associated with the transition process and lead to a more dynamic environment in which an end-system observes rapid and marked fluctuations in network characteristics [8]. There is, therefore, an emerging requirement for a new class of distributed system service designed specifically to operate in this dynamic environment. Such services must avoid assumptions about their underlying support environment which prevents them from operating effectively across a range of networks. We term this new class of service *adaptive services*.

The potential of adaptive services is considerable. Firstly, they can fully exploit the available level of connectivity at any given time. Secondly, adaptive services are more portable. For example, the same service could be used on a workstation connected to a high speed network and on a mobile computer in the field.

Adaptive services differ from conventional distributed system services in their ability to exploit changes in the quality of service of the underlying communications infrastructure. It is therefore important to provide facilities which enable these applications to monitor and manage the QoS provided by the underlying network.

3: A Distributed Systems Platform to Support Adaptive Services

3.1: The ANSAware Distributed Systems Platform

The implementation of our platform is based on APM Ltd.'s ANSAware software suite. This software suite is itself based on the ANSA architecture which has had a profound influence on the RM-ODP. Thus, the platform tackles the problem of developing applications to operate in a heterogeneous environment. The ANSA programming model is based on a location-independent object model where all interacting entities are treated uniformly as encapsulated objects. Objects are accessed through operational interfaces which define named operations together with constraints on their invocation. Objects are made available for access by exporting interfaces to a special object known as the *trader*. An object wishing to interact with this interface must then import the interface from the trader by specifying a set of requirements in terms of a interface type and attribute values. This will be matched against the available services and a suitable candidate selected. At this stage, an implicit *binding* is created to the object supporting the interface, i.e. a communication path is established to the object. Invocation of operations can then proceed.

To provide a platform conformant with the above programming model the ANSAware suite augments a general purpose programming language (usually C) with two additional languages. The first of these is IDL (Interface Definition Language), which allows interfaces to be precisely defined in terms of operations, arguments and results. The second language, DPL (Distributed Processing Language) is embedded in a host language, such as C, and allows interactions to be specified between programs which implement the behaviour defined by these interfaces. Specifically, DPL statements allow the programmer to import and export interfaces, and to invoke operations in those interfaces.

In the engineering infrastructure, the binding necessary for invocations is provided by a remote procedure call protocol known as REX (Remote EXecution protocol) or a group execution protocol know as GEX (Group EXecution Protocol). These are layered on top of a generic transport layer interface known as a *message passing service* (MPS). A number of additional protocols may be included at both the MPS and the execution protocol levels and these may be combined in a number of different configurations. The infrastructure also supports lightweight threads within objects so that multiple concurrent invocations can be dealt with.

All the above engineering functionality is collected into a single library, and an instance of this library is linked with application code to form a *capsule*. Each capsule may implement one or more computational objects. In the UNIX operating system, a capsule corresponds to a single UNIX process. Computational objects always communicate via invocation at the conceptual level but, as may be expected, invocation between objects in the same capsule is actually implemented by straightforward procedure calls rather than by execution protocols. ANSAware currently runs on a variety of operating systems platforms including various flavours of UNIX, VMS and MS-DOS/Windows.

3.2: Extensions to Support Multimedia and Mobility

We have extended the basic ANSAware platform to support the transmission of continuous media and operation in a mobile environment. In particular, we have significantly extended the support for bindings which support both of these new facilities. The current version of ANSAware (version 4.1) supports only implicit bindings, i.e. a client who obtains a reference to an operational interface may use that reference without an explicit bind statement. The example in figure 1 shows a client importing a 'stack' service and subsequently invoking an operation on that service's interface.

Failures in the implicit binding carried out by the system are reported at invocation time (typically the first invocation).

---

1 The MINT router is being developed as part of the Walkstation project [Hager,93] and enables mobile computers to dynamically exploit the services offered by a number of networks. In particular, it supports seamless transitions between network types.

```
! {stack_control} <- traderRef$Import
      ("Stack", "context", "properties")
! {result}<-stack_control$Push (value)
```

Figure 1 : Interface Interaction in ANSA

In order to communicate continuous media information additional support is required: a continuous media communication cannot be represented as a *single* invocation because of its potentially unbounded nature [Coulson,92]. For example, the output from a continuously running surveillance camera cannot be captured as the result of a single invocation. Moreover, continuous media communication cannot be represented as a *sequence* of invocations because it is not possible to specify synchronisation constraints (e.g. jitter) that apply to a sequence of invocations.

To meet the requirement for an abstraction of real-time data flow over time, we have added the concept of *explicit stream bindings* to our platform. Stream bindings provide an end-to-end abstraction over continuous media communication and support arbitrary m:n connections, i.e. they allow m sources to be connected to n sinks.

Continuous media sources and sinks are represented as objects with *stream interfaces* (c.f. standard or *operational interfaces*). These interfaces contain a specification of a set of one or more flows including the media type (e.g. audio), the direction (in or out) and the associated QoS specification (see figure 2). The QoS specification can be used to state, for example, the required throughput, latency and jitter characteristics of a binding to the interface. A stream binding between two stream interfaces may thus comprise of a number of component flows with differing QoSs and communicating information in different directions.

```
Telephone: STREAM_INTERFACE =
BEGIN
              FLOW_SPEC 1 AUDIO IN        64,
20, 5, 10
              FLOW_SPEC 2 AUDIO OUT       64,
20, 5, 10
END.
```

Figure 2 : An example stream interface

Within our platform stream bindings are established using an explicit bind operation (see figure 3) which takes as parameters the source and sink interfaces to be bound and a further set of parameters which express the desired QoS . More details of the specification of QoS parameters for continuous media bindings can be found in [10].

```
! {src}<-traderRef$Import
      ("Telephone","context","properties")
! {sink} <- traderRef$Import
      ("Telephone","context","properties")
! {binding_control_ir} <- binder$Bind
      (Stream,src,sink,QoS)
```

Figure 3 : Creating an explicit binding

Clients are returned a binding control interface as a result of an explicit bind operation (see figure 4). To conform to the RM-ODP all explicit binding control interfaces must support at least the operation *unbind*, but in addition the control interfaces for our stream bindings contain a number of additional operations, e.g. to connect and disconnect sources and sinks. To control the QoS of the stream once the binding has been established the control interface includes a pair of operations *setQoS()* and *getQoS()*. These operations take as arguments a set of QoS parameters which can then be passed by the stream binding to the underlying transport protocol. A call-back mechanism is also provided to inform client objects of QoS degradations reported by the underlying transport service.

Multicast and multidrop bindings can be established by supplying grouped stream interfaces (groups of interfaces are a standard part of the RM-ODP) as parameters to the bind operation. Note that we do not support the connection of additional source and sink interfaces to a binding once it has been established. This is because we wish to model groups as explicit first class objects rather than objects created as a side-effect of a binding. Source and sink interfaces can however be added dynamically to a bound group thus the interfaces involved in a binding may change during its lifetime. Further details on both modelling and engineering aspects of stream bindings and groups can be found in [9, 10, 11].

```
BindingControl : INTERFACE =

BEGIN
   Callback : TYPE = {
      QoSViolationCallback,
      ClientMemberPolicy,
      ServerMemberPolicy };

   Unbind :
   OPERATION   [ ]
   RETURNS     [ ];

   SetQoS :
   OPERATION   [ NewQoS : QoS ]
   RETURNS     [ Status, QoS ];

   GetQoS :
   OPERATION   [ ]
   RETURNS     [ QoS ];

   Register :
   OPERATION   [ Callback_Type :
                 Callback;
                 Callback_Interface :
                 Interface ]
   RETURNS     [ Status ];

   DeRegister :
   OPERATION   [ Callback_Type :
                 Callback;
                 Callback_Interface :
                 Interface ]
   RETURNS [ Status ];

END.
```

Figure 4 : IDL specification of the binding control interface

In addition to stream bindings for the transmission of continuous media we have also added a new class of explicit binding for use with operational interfaces. These bindings are established using the binder$Bind operation as above but take as arguments operational interfaces. The resulting binder control interface is identical to that used for stream bindings except that clients are allowed to specify and monitor a different set of QoS parameters associated with the binding. This enables, for example, a client to ask to be informed when the QoS supplied by the binding falls below a specified threshold. Of particular relevance to mobile applications is the ability to monitor the possibility of sending or receiving messages via a specified binding without having to explicitly send application level test messages, i.e. applications can delegate responsibility for guaranteeing QoS assertions to the system. This is of significance since it allows mobile applications to be structured in an event based fashion (c.f. polling). For example, through the use of our bindings it is possible to assert that the absence of messages on a given interface is a result of there being no traffic intended for the specified interface rather than a result of communications failure. In addition, QoS driven bindings allow the system to optimise the use of test messages which might otherwise be duplicated if left to individual applications, e.g. if multiple applications wished to test QoS assertions between the same pair of objects. Further details of the structural changes in applications and system services possible as the result of the introduction of QoS driven bindings can be found in [6]).

3.3: Implementation Experiences

During our initial attempts to implement the above facilities we have encountered problems with the ANSAware remote procedure call protocol REX. In particular, REX takes no account of the characteristics of the underlying network; parameters such as the number of retry attempts and the interval between these attempts are fixed at installation time. Thus, when a system configured to operate over an Ethernet is run over a low-speed network the absence of congestion control within REX means that almost no data is actually communicated between user processes (the absence of congestion control is not the only problem with REX; indeed, RPC protocols with congestion control have been shown to perform poorly over low-bandwidth communications links [12]). Instead, the network becomes overloaded with REX control messages. Note that while different transport protocols can be placed underneath REX the execution protocol does not take account of any feedback such protocols might provide regarding (for example) congestion control. Similarly, the GEX protocol is wholly unsuitable for use over many wireless networks. The protocol has been designed to avoid a central point of failure for group communication. Instead, group events such as membership changes and message arrivals are co-ordinated by all group members exchanging state information using an internal token-based protocol. This generates redundant messages if hardware broadcast is supported, and involves the establishment of multiple point-to-point connections between group members if hardware broadcast is not supported.

We are therefore in the process of developing a number of new protocols. The first of these has been in operation for some time and supports the transmission of continuous media via stream bindings. The protocol optimises mixing of continuous media by re-locating the mixing process based on the binding configuration. In order to support explicit bindings between operational interfaces we are developing a pair of new execution protocols called QEX (Quality-of-service driven remote EXecution) and G-QEX (Group-Quality-of-service driven remote EXecution). These protocols will gather information on specified bindings based on (for example) the number of retries and average delay they experience over a given channel and be able to pass this information on to interested clients. This will allow our new protocols to implement congestion control *and* to propagate this information to applications in order that they can 'back-off' to further reduce network traffic. In the case of G-QEX we hope to be able to adjust the protocol's approach to supporting groups (i.e. centralised or distributed) based on network and user QoS information. Where necessary QEX and G-QEX will periodically test bindings in order to be able to guarantee QoS assertions.

4: A Prototype Adaptive Application

In order to evaluate the distributed systems platform described in section 3 we have implemented a prototype adaptive application. The application is designed to support field engineers within the utilities industries and has two novel features: it provides explicit feedback to users on the state of the communications channels available to the application, and, it supports *collaboration* between mobile users. In more detail, the application allows field engineers to:-

(i) View and manipulate maps and circuit diagrams on their mobile computer using a customised geographic information system (GIS).

(ii) Establish conferences of engineers with support for audio communication between participants.

(iii) Exploit the functionality of the GIS in a conference setting, i.e. show and manipulate maps and diagrams to all or a subset of the conference participants. Highlighting or 'Red-Lining' of the images is also permitted by the GIS.

We have structured the application as a series of modules (e.g. a GIS module) all of which communicate using our distributed systems platform. The modules are controlled using a single control window on each mobile computer which allows users to start and stop modules on both their own and remote machines and to establish and manage conferences. This single control window (shown

top left in figure 5) is also used to provide users with feedback on the state of their communications channels. This is achieved by changing the colour of icons representing remote applications and other conference participants based on feedback from the distributed systems platform. This allows users to modify their behaviour to improve application performance.

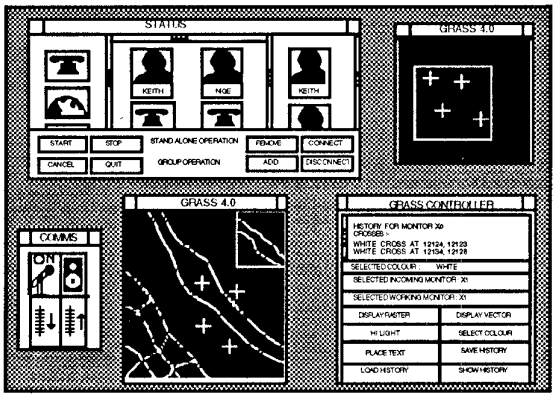

Figure 5 : The User Interface

Currently our application runs on a mixture of notebook and desktop PCs and Sun workstations connected via Ethernet. The application will be used in this environment when engineers spend time in the office. When the development of QEX and G-QEX is complete we be able to test the application in the field using low-bandwidth wireless communications.

5: Concluding Remarks

This position paper has presented a rationale for adaptive services in a heterogeneous mobile environment and described a distributed systems platform featuring support for such services. Ongoing research is using this platform to develop applications including a multimedia conference utility to support electricity workers in the field and a file system, based on Coda [13], which can operate in a range of network environments.

Perhaps the most important issue emerging from this work is need for new protocols (for both remote execution and group execution) which are flexible enough to operate over a range of networks and which are capable of providing feedback to applications on the quality of service provided by the underlying communications infrastructure thus enabling *both protocols and applications* to react to changes in QoS.

Acknowledgements

The work described in this paper was carried out as part of the SERC/DTI funded MOST (Mobile Open Systems Technologies) project.

References

[1] Duchamp, D. "Issues in Wireless Mobile Computing." Proc. Third Workshop on Workstation Operating Systems, Key Biscayne, Florida, U.S., 1992. IEEE Computer Society Press, Pages 2-10.

[2] Katz, R.H. "Adaption and Mobility in Wireless Information Systems." IEEE Personal Communications Vol. 1 No. 1, Pages 6-17.

[3] ISO. "Draft Recommendation X.901: Basic Reference Model of Open Distributed Processing - Part1: Overview and Guide to Use", Draft Report 1992.

[4] OSF. "Distributed Computing Environment: An Overview", Open Software Foundation. April 1991.

[5] OMG. "The Common Object Request Broker: Architecture and Specification (CORBA)", Report 91.12.1, The Object Management Group. 1991.

[6] Davies, N., S. Pink, and G.S. Blair. "Services to Support Distributed Applications in a Mobile Environment." Proc. 1st International Workshop on Services in Distributed and Networked Environments, Prague, Czech Republic, June 1994.

[7] A.P.M. Ltd. "The ANSA Reference Manual Release 01.00", APM Cambridge Limited, UK. March 1989.

[8] Hager, R., A. Klemets, G.Q. Maguire, M.T. Smith, and F. Reichert. "MINT - A Mobile Internet Router." Proc. IEEE VTC'93, Secaucus, NJ, U.S., 1993.

[9] Coulson, G., G.S. Blair, N. Davies, and N. Williams. "Extensions to ANSA for Multimedia Computing." Computer Networks and ISDN Systems Vol. No. 25, Pages 305-323.

[10] Coulson, G., G.S. Blair, F. Horn, L. Hazard, and J.B. Stefani. "Supporting the Real-Time Requirements of Continuous Media in Open Distributed Processing." To Appear in Computer Networks and ISDN Systems.

[11] Davies, N., G. Coulson, N. Williams, and G.S. Blair. "Experiences of Handling Multimedia in Distributed Open Systems." Proc. 3rd USENIX International Symposium on Experiences with Distributed and Multiprocessor Systems, Newport Beach, CA, U.S.A., March 1992.

[12] Bachmann, D., P. Honeyman, and L.B. Huston. "The Rx Hex." Proc. 1st International Workshop on Services in Distributed and Networked Environments, Prague, Czech Republic, June 1994. IEEE Computer Society Press, Pages 66-74.

[13] Kistler, J.J., and M. Satyanarayanan. "Disconnected Operation in the Coda File System." ACM Transactions on Computer Systems Vol. 10 No. 1, Pages 3-25.